6/2/2016

BatView® Sonar Viewer

8360359

Ryan Ewyk

Contents

Summary 2

Design Document 2

Programming Research Log 3

Code Description 4

Code 5

Summary

The BatView® Sonar View is the interfacing and usage of the four ultrasonic HC-SR04 sensors on the SART robot. The goal of this script is to collect and process the data given from the 4 ultrasonic sensors and process it, ready to send to the webserver which will generate the 2D map of the surrounding area of the robot. The program is also responsible for controlling the Dynamixel servos which control the movement of the robot. To do this, it constantly checks for input from the servos (through serial) and runs scans using the four ultra-sonic sensors. To scan using the ultrasonic sensors, the Arduino board simply outputs a digital signal to the TRIG pin for ten milliseconds and then collects the eight responses from the ECHO pins and converts outputs into distance. When sent over serial to the PI, the data is given a letter ('a', 'b', 'c' or "d") based on which sensor it came from. This allows the PI to sort the data into what data came from which sensor to build an appropriate map.

The program will also gather data from a magnetometer and accelerometer which will also be sent to the Raspberry Pi, so that relative orientations can be accurately determined in order to generate the map.

Design Document

The program gathers data from three sources: Four HC-SR04 ultrasonic sensors, four Dynamixel 12-A servos and the Raspberry Pi. It needs to send and receive data to and from the ultrasonics in order to calculate the distance between them and the robot. The Dynamixel servos are written to over the serial 2 (pins 18 and 19) in order to control the speeds of which the servos run at. Lastly, the program needs to read and write to the Raspberry Pi using serial. It reads incoming data in order to control the servos from the Pi, but it also needs to write the collected distance data from the ultrasonics.

```
#define PRESENT_POS_L 0x36
#define PRESENT_POS_H 0x37

#define echoPin_1 11 // Echo Pin
#define trigPin_1 12 // Trigger

#define echoPin_2 11 // Echo Pin
#define trigPin_2 12 // Trigger Pin

#define echoPin_3 11 // Echo Pin
#define trigPin_3 12 // Trigger Pin

#define echoPin_4 11 // Echo Pin
#define trigPin_4 12 // Trigger Pin
```

Figure 1

The program needs to store and manipulation many different forms of data in order to achieve its functional goals. The first thing that is done is created global constants for all of the pins being used, as well as two constants needed for the servos (figure 1). These are very important constants that are stored to keep track of the pin setup of the board. After that two variables are created to store longs. They will store the "duration" (How long it took to get a response when the TRIG is triggered) and "distance" (The distance between the sensor and the object the sensors detected in cm). Next a function is created to process the incoming bytes from the Raspberry Pi, which is simply reading them. The next function deals with those bytes, checking if they fit any of the requirements for processing (Explained in more depth in the Code Description). Based on what is received, it is determined what mode which motor should be set to, as well as what speed. Triggering and receiving ultrasonic echoes are dealt with in the next function *getDistanceFrom* as well as converting the inputs into distance (cm). The function then returns the calculated distance, returning "-1" if the distance is further away than the maximum distance. The function

convertToString, is all about data manipulation, as it converts an integer into a string so it then can be further processed in the next function, so it can be sent to the Raspberry Pi. In the *loop()* function, first function is used to get the bytes. It is here that it is determined whether the information is telling the servos to be written to or read from. It uses a switch function to determine if the first byte reads "W" (write) or "R" (read). Directly after this process, the four ultrasonic sensors are run using the previous *getDistanceFrom()* function and converted into strings with the *convertToString()* function. When the inputs have been converted a letter is added to the end, to determine which sensor it can from. The last function, requires an input of a number, which it then processes, setting the inputted variables pointer to fit the other incoming bytes.

Programming Research Log

I used the Robotis™ Open CM IDE to write my program. Since the board and am writing to is not exactly an Arduino board, the developers created its own IDE to write programs to it. It is very similar to Arduino, not only having most if not all of the libraries from the Arduino IDE, but also very similar GUI and functionality, making it easy to move from Arduino (which is what I was using before I switched) to Robotis's IDE.

Probably one of, if not the most useful programming technique I have used is to group statements in their own functions for easier use and reusability, efficiency and overall readability of the code. The functions I have written are used throughout the program, usually more than twice. Within some of these functions I have used switch statements. When handling the processing of which incoming byte does what, using a switch statement is far more efficient that using if-else statements for the program doesn't need to keep checking conditions; it can simply skip everything that doesn't apply to the bytes and execute what it needs to. The use of loops was also very useful in my code as they allow compact and again efficient ways to do certain things. I used a while loop to check if there was incoming bytes while the serial ports were active, as well as a for loop to loop through the size of a short to store in the buffer array for reading the incoming bytes over the serial.

I used a couple of online tutorials with one sample application from the Robots Open CM IDE. I used one tutorial to help me with interfacing with the ultrasonic sensors and another website to help with setting up the ultrasonic sensor for testing and algorithmic ideas for interfacing with the sensors. I used the sample application from the Open CM IDE to help me with using the serial library.

Interfacing with the ultrasonics - http://arduinobasics.blogspot.com.au/2012/11/arduinobasics-hc-sr04-ultrasonic-sensor.html

Electronic setup for testing and algorithms - http://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi

Open CM IDE sample application - http://support.robotis.com/en/software/robotis_opencm/learn/serialcomunication.htm

I used mainly two libraries in my program. They were necessary in order to get the program functional, as they were mainly used to interface with the physical components.
Serial – I used the serial library which was a default library that came with the Open CM IDE. This allowed me to interface directly with the serial ports of the Open CM board so I can send and receive data through them.
Dynamixel – I used the Dynamixel library which also came with the Robotis™ IDE. Using this library, controlling the Dynamixel servos was easy and gave me full control over the many feature they provided.

Despite the simplicity of the program now, I encountered many problems. The biggest problem I had to deal with for this project was working with Arduino. I had no history working in Arduino before this project, so I had to learn the basics of Arduino. It was because of this that I had a problem with libraries, I needed a library for the Dynamxiel servos but original Arduino IDE did not have it. Another significant problem I had encountered was figuring out to use serial to communicate to the pi. To do this, I learnt how serial communication works on the physically and how it would be implemented on the robot. Once I knew how it would work physically, I could implement the serial library to interface with it.

Code Description

It starts off initialising constants for pins, both trig and echo, as well as the initialising the dynamixel library. Two variables are also initialised, the distance and duration of signals from the ultrasonics. Next I wrote a function to setup all of the ultrasonic pins. The next function is the Arduino setup function which setups the serial and serial 2 ports, as well the dynamxiel servos and the ultrasonic setup. The *blockingRead()* function reads from the serial ports to collect the data from the pi to control the servos. But this is not enough. The *readInt()* function acts as a buffer for this process. In order to actually use the commands that were sent from the pi, they need to be processed. The *processWrite()* function sorts the data in order to make the servos do the right thing that was requested from the Pi. For example, if a "J" is sent to the Open CM board, this function will sort it and make the desired servo go into joint mode. The *loop()* function is where all of the functions are used together to make the program functional and achieve its objective. The first half of this function is sorting whether the first byte being read is an "R" for reading or "W" for writing and appropriately by writing using the *processWrite()* function. After focusing on the servos the function now deals with the ultrasonic sensors. It uses 'serial.write()' to send the distance in string form with the letter identifier to the Raspberry Pi. To get the distance, the function getDistanceFrom() is called. This function has two parameters, being a trigger pin and an echo pin. The function sends a digital signal to the trig pin on the sensor for 10 microseconds to get 8 pulses back (or one byte). This information is then received using the pusleIn() function. This data is stored in the duration variable. After that is divided by 58.2 to get the distance in centimetres. It then returns the data, but it only returns the processes data if it is less than 380cm (3.8m). If it is not less than that, it returns -1. To turn that returned data into a string with its letter identifier, the function *convertToString()* is used. This uses a trick that initialise a new string but uses the number as the parameter to turn it into the string. It is then returned with the identifier add on last. The program loops through that function continuously thus allowing the program to complete its task to constantly update the servos and gather data from the ultrasonic sensors.

Code

```
Dynamixel       Dxl(1);                 // Initilise Dynamixel library

#define echoPin_1 11           // Echo Pin a
#define trigPin_1 12           // Trigger a

#define echoPin_2 11           // Echo Pin b
#define trigPin_2 12           // Trigger Pin b

#define echoPin_3 11           // Echo Pin c
#define trigPin_3 12           // Trigger Pin c

#define echoPin_4 11           // Echo Pin d
#define trigPin_4 12           // Trigger Pin d

long            duration,
                distance;      // Duration used to calculate distance

    // Function for the ultrasonics pin setups
    void
ultra_setup()
{

pinMode(trigPin_1, OUTPUT);

pinMode(echoPin_1, INPUT);

pinMode(trigPin_2, OUTPUT);

pinMode(echoPin_2, INPUT);

pinMode(trigPin_3, OUTPUT);

pinMode(echoPin_3, INPUT);

pinMode(trigPin_4, OUTPUT);

pinMode(echoPin_4, INPUT);


}
    // Arduino 'setup' function
    void
setup()
{

Serial.begin(9600);    // start up 'serial' pins 0 and 1
    Serial2.begin(57600);      // Stat up 'serial 2' pins 18, 19
    Dxl.begin(3);              // setup 4 dynamixel
    ultra_setup();             // Call the ultrasonic setup
}
```

```
    // function for reading the incoming bytes from the 'serial 2' or
    // Raspberry Pi
    byte blockingRead()
{

while (!Serial2.available())
        delay(10);              // Check if serial 2 is available
    return Serial2.read();      // return the data from the serial 2
}


    // Function to write data to the Dynamixel servos using the data from
    // receieved from Raspberry Pi
    void
processWrite(byte cmd, byte id)
{

switch (cmd)

 {

case 'j':                       // if 'j' has been sent
        Dxl.jointMode(id);      // make the desired servo a 'joint'
        break;

case 'p':                       // if a 'p' was sent
        int             pos,
                        vel;    // make two variables, one for position
                                // and one for velocity
        if (!(readInt(&pos) && readInt(&vel)))
            break;              // If they don't dont exist break out of
                                // switch
        Dxl.setPosition(id, pos, vel); // if so, set the position of the
                                       // desired servo with the position
                                       // and velocity
        break;

case 'w':                       // if a 'w' was sent
        int             add,
                        val;    // make to variables, add for the goal
                                // position and val for the position
        if (!(readInt(&add) && readInt(&val)))
            break;

Dxl.writeWord(id, add, val);    // make the desired servo move at
                                       // the specified goal position and
                                       // position
        break;

}

}

    // Function that gets the distance of an object detected by the
    // ultrasonic sensors, returns the distance in cm
    int
```

```
getDistanceFrom(int trig_pin, int echo_pin)   // has two parameters, a
                                              // trig pin and an echo
                                              // pin
{

digitalWrite(trig_pin, LOW);   // send a 'low' digital signal to
                                       // the trig pin
    delayMicroseconds(2);      // wait 2 microseconds

digitalWrite(trig_pin, HIGH);  // send a 'high' digital signal to
                                       // the trig pin
    delayMicroseconds(10);     // wait 10 microseconds

digitalWrite(trig_pin, LOW);   // send a 'low' digital signal to
                                       // the trig pin
    duration = pulseIn(echo_pin, HIGH);       // set the variable
'duration' to
                                       // the returned value from the
                                       // 'pulseIn' function

        // Calculate the distance (in cm) based on the speed of sound and
        // return it
        return duration / 58.2 > 380 ? duration / 58.2 : -1;


}

    // function for converting integers into strings
    string convertToString(int number, string id)     // takes two
                                                      // parameters: the
                                                      // number to be
                                                      // converted and
                                                      // the id of the
                                                      // ultrasonic
{

String convert = new String(number)    // puts the number into a
                                              // string
        return convert = convert + id  // returns it with the id added on
                                       // to it
}


    // function that loops continuously
    void
loop()
{

byte inByte = blockingRead();  // get the byte thats coming
                                       // through
    switch (inByte)

 {

case 'W':                      // if the byte stream starts with 'w'
        {
```

```
byte cmd = blockingRead();     // get the next two parts: the
                                    // command
            byte            id = blockingRead();      // and the id (of
                                                      // the servo)
            processWrite(cmd, id);     // Process the commands with their
                                    // ids
        }

break;

case 'R':

            // byte ID = blockingRead();
            // readFromDynamixel(ID);
            break;

}


Serial.write(convertToString(getDistanceFrom(trigPin_1, echoPin_1)), "a");
        // Write

                // the

                // distance

                // from

                // ultrasonic

                // 'a'

                // to

                // the

                // pi
    Serial.write(convertToString(getDistanceFrom(trigPin_2, echoPin_2)),
"b");   // Write

        // the

        // distance

        // from

        // ultrasonic

        // 'b'

        // to

        // the

        // pi
    Serial.write(convertToString(getDistanceFrom(trigPin_3, echoPin_3)),
"c");   // Write
```

```
        // the

        // distance

        // from

        // ultrasonic

        // 'c'

        // to

        // the

        // pi
    Serial.write(convertToString(getDistanceFrom(trigPin_4, echoPin_4)),
"d");   // Write

        // the

        // distance

        // from

        // ultrasonic

        // 'd'

        // to

        // the

        // pi

        //

}

    // function to read the incoming bytes
    boolean readInt(int *outVal)
{

byte inBuffer[sizeof(short)];  // make buffer array - hold 2
                                        // bytes
    int             i;          // counter integer
    for (i = 0; i < sizeof(short); i++)       // loop through the size of a
                                        // short (2 bytes)
    {

inBuffer[i] = blockingRead();  // put the bytes into the buffer
                                        // array in with the counter
    }
*outVal = *((short *) inBuffer);        // set the parameter to point to
                                        // the array
    return true;                // return true
}
```